Bachelor's thesis

Degree programme in Information Technology

NTIETS13

2017

Adomas Simonenkovas

CONTINUOUS INTEGRATION FOR A CUSTOMER MANAGEMENT SERVICE SOFTWARE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology

2017 | 41

Adomas Simonenkovas

CONTINUOUS INTEGRATION FOR A CUSTOMER MANAGEMENT SERVICE SOFTWARE

CI (Continuous Integration) is a practice used in extreme programming environments to ensure the integrity and quality of the code in a version control system. The main tasks of CI are retrieving the newest version of the code and organizing application dependencies, running unit and static tests tools, and packing the code into a version that can be deployed to a server or to a customer.

The purpose of this thesis was to find out and implement the best suitable solution for the project being tested, which was a PHP application.

A meeting with the commissioned was held to determine the subject and the requirements of this thesis followed by a revision of literature on software testing and continuous integration. The materials used in researching varied from literature to software documentation. The scope of this thesis range from server installation which requires knowledge of Linux server environments to configuring projects in CI which require the understanding of XML format which is a file format usually used for the settings of an application.

Research carried out in this thesis revealed that continuous integration is really simple, fast to set up and simplifies the process of software testing during the software development life cycle. It has open source build solutions for different applications and can be modified widely to meet any company requirements.

Implementing CI to the project revealed a few coding standard violations such as the naming of classes, methods, variables and availability of methods which can be interpreted as security defects. Test cases and scrips are still required to be implemented to ensure that the application works correctly.

KEYWORDS:

Continuous Integration, Jenkins, Version control, Software testing

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma

2017 | 41

Adomas Simonenkovas

JATKUVA INTEGRAATIO TIETOHALLINTAJÄRJESTELMÄLLE

Jatkuva integraatio on prosessi, joka on yleisesti käytetty extreme programming -ympäristössä. Jatkuva integraatio takaa sen, että ohjelmistonkehittäjän tuottama koodi rakentuu toimivaksi ja että se on korkealaatuista. Jatkuvan integraation tehtäviin kuuluuvat ohjelmakoodin noutaminen versiohallintajärjestelmästä, riippuvien kirjastojen hoitaminen sekä yksikkö että staattisten testien suoritus ja ohjelmakoodin pakkaus muotoon, jonka voi toimittaa suoraan palvelimelle tai asiakkaalle.

Opinnäytetyön tarkoituksena oli suunnitella ja implementoida paras ratkaisu testattavaa projektia varten, joka oli php-sovellus.

Opinnäytetyön aihe ja vaatimukset laadittiin yhdessä toimeksiantajan kanssa palaverissa. Opinnäytetyö aloitettiin ohjelmistotestauksen ja jatkuvan integraatio-aiheisten kirjojen lukemisella. Opinnäytetyössä käytettiin lähteinä kirjallisuudesta ohjelmistodokumentaatioon. Opinnäytetyössä tutkittiin testipalvelimen asentamisesta, joka vaatii tietämystä Linux-palvelinympäristöstä ja projektien konfigurointiin jatkuvassa integraatiossa. Projektien konfigurointi vaati ymmärrystä XML-muotoisten tiedostojen lukemisesta, joka on yleisesti käytetty tiedosto muoto sovellusten asetusmäärittelyjä varten.

Tutkimus osoitti, että jatkuva integraatio on helppo ja nopea asentaa, sekä se helpottaa ohjelmiston testausta ohjelmistokehitysvaiheiden aikana. Verkossa on monia avoimen lähdekoodin ratkaisuja, joita voi käyttää monessa eri sovelluksessa tai niitä voi muokata yrityksen tarpeen mukaan.

Jatkuvan integraation soveltaminen annettuun projektiin paljasti muutamia virheitä koodi standardeissa, kuten luokkien, muuttujien ja metodien nimeäminen, sekä metodien tyypit, jotka vaikuttavat sovelluksen tietoturvaan. Testitapausten ja skriptien laatiminen on vielä tehtävä, jotta ohjelma toimii luotettavasti oikein.

ASIASANAT:

Jatkuva Integraatio, Jenkins, Versionhallinta, Ohjelmistontestaus

CONTENTS

FIGURES

LIST OF ABBREVIATIONS (OR) SYMBOLS

Abbreviation	Explanation of abbreviation (Source)
CI	Continuous Integration
XML	Extensive Markup Language

1 INTRODUCTION

Coding projects have evolved a lot during the past decade in several ways. Development teams have grown separating the location of teams different parts of a country or even a continent, making daily communication harder within the organization. New software development models such as Agile methods have taken over the old practices such as the waterfall structure. Waterfall model gets its name by aligning the steps in a stair shaped formula resembling a waterfall and follows 5 stages. The 1st stage is gathering and defining the requirements for the software being developed. This stage is done between the project manager and the customer. A failure in determining the correct requirements has a high risk of failing the whole project. The 2nd stage is designing the software. This is separated in to two subphases, logical and physical design. Logical designing phase means that the system is designed independently of hardware or software based on the requirements gathered during the first phase. Physical designing is done after logical and is dependent on hardware and software. The 3rd stage is implementation, which means the software is developed according to the requirements and specifications. Verification is the 4th phase of the waterfall model. This phase is ment to verify if the project still meets the customers needs. The last stage is maintaining the software. During the maintaining stage the customer is already using the software. Changes are made In this stage as problems are found in poor requirements, mistakes in the previous stages or the change of customer's requirements.

Following the waterfall model has both advantages and disadvantages. Advantages include excellent documentation, helping new developers to understand the system. It is well structured helping the track progress of the project determined by milestones. The costs of the project can be accurately calculated after requirements have been defined. Disadvantages include the difficulty of defining correct requirements therefore failing the whole project in early stages, project's requirements can't be changed during the development cycle and the delivery of the project can take longer. (Hughey 2009)

Alternative to waterfall model is agile methods such as scrum. Scrum is the most common agile framework. It is common in software projects, since it adabtable to changes of requirements. There are 3 roles in scrum: product owner, scrum master and development team. Product owner is the person who the software is developed to and knows the requirements. Scrum master is normally a person from the development team,

who plans the upcoming development period also known as a sprint and supports the development team. Sprints can last normally from 1 to 4 weeks. The project as a product backlog, which has components that the software needs. Sprint backlog are the items the team is commited to develop during the following sprint. After the sprint backlog is agreed, development of the components can begin. The idea of this framework is to chop down the project into small parts and develop them in short periods instead of developing the whole software in one go and presenting it to the customer. If the customer requirements change, it is added to the product backlog and developed when chosen to the sprint backlog.

Most companies use some sort of agile framework, which is a mixture of scrum and waterfall model having both components in use. Using an agile development method requires dividing the software into smaller units, which are normally developed by several people simultanuously. Developing the same software simutanuously can break the software or cause a conflict, which means that developers edit the same code without knowing it and causing it to break. How can this all be tested so every new commit of code results a working version of the software?

One solution would be having testers deploy and buil the software on a testing environment, run all the tests and check the code each time a new version has been committed to the source code management system. This however would cause the costs of the testing team to skyrocket.

Another possibility would be to setup a system, which retrieves the new source code, deploys and builds the software on a test environment and runs the determined tests, when a newer version is available on the source code management. This can be handled by continuous integration tool such as Jenkins, which will be explained later on in the thesis.

The source material used for the thesis will be a book on testing and continuous integration, api documentations of the used tools and systems and also other thesis, that are related to continuous integration and test automation. The book and the api documentations will be used as trusty sources since they are genuine and independent. Other thesis will be used to gather ideas and compare them to the trusty sources.

2 THEORY

This chapter will cover the theoretical part of the thesis. It contains the general concepts of Version Control, Continuous Integration and Software Testing that is being performed by Continuous integration. This chapter's intention is to give the basic knowledge of different components that are a part of continuous integration. After reading this chapter the reader should understand what is version control and continuous integration and how it works. This chapter will also cover the importance of software testing.

2.1 Version Control

Version Control System allows a team of software developers to work on the same project simutanously. It creates a "database" for the source code, creating a backup of the code everytime a developer commits a change to the source code. This enables the developers to check out the newest version of the source code at any given moment, compare their version of the code to the remote repository to see the changes they have made to the code and revise the code back to a last working state in the case of an error. The version control system should save the action being performed to the source code, the author, date of the commit and commit notes, which explain what was changed. Version control is crucial in software development because:

- It saves the earliest versions of the source code, dating several years.
- It enables developers to "fork" their work into different branches, which don't effect each other.

Version Control Systems can be devided into three different categories; Local Version Control System, Centralized Version Control System and Distributed Version Control Systems. This thesis will focus mainly on Distributed Version Control Systems, since git is used for this project.

Local version control system is the act of copying the files to a new folder and naming the folder with the creation date. This method does create different versions of the files, but does not provide any information what was changed or added and the access is restricted to local access only.

Centralized Version Control System

2.1.1 Git

Git is a distributed version control system, which means that everytime a developer checks out the repository, a clone of every version of the source code is also made (see figure 1.). As seen from the figure the server has has each version of the code. Developers have the same database of the server from the time it was cloned into the developers computer. As seen in this situation, if the server crashes and loses its database, any of the developers can restore the server's database as their own clone work as a backup.

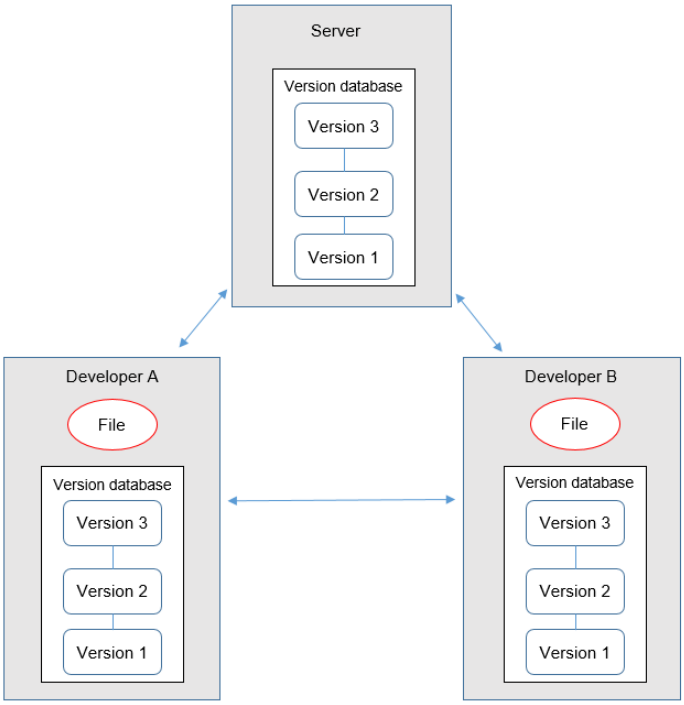

Figure 1. Distributed version control Hierarchy (Getting Started - About Version Control)

2.2 Continuous Integration

"Continuous Integration is designed to create an automation environment to ensure that every change made to the application code results a releasable version of the application." The criteria if the results are releasable are determined during the configuration part of the CI software. The goals of CI is to make the tradionationally

manual software development processes such as building the application, running unit tests and code analysis. (Karadzhov 2013)

The usual Continuous Integration workflow follows this pattern (see figure 2 below):

1. The developers commit/push their local version to the remote source code repository.
2. The branches are merged in to the master branch.
3. The Continuous Integration system monitors changes made to the version control system and triggers a build when a newer version of the code is available.
4. The Continuous Integration server runs the unit tests.
5. The Continuous Integration server runs the static tests.
6. Continuous integration server deploys the software on a test server
7. Continuous Integration Server runs integration tests
8. If any steps from 4 to 7 fail, the developers are informed that the build failed. Otherfice the release team if mentioned that the version is releasable. (James 2013)

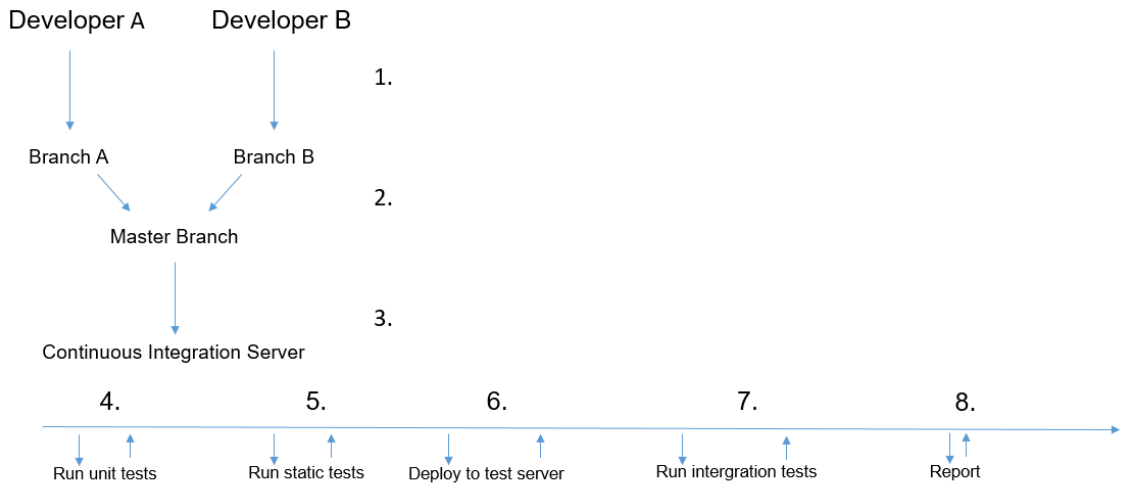

Figure 2. Continuous Integration workflow (James 2013)

There are various benefits in having Continuous Integration as a part of software development:

1. Eliminates the need for integration testing, which would require time from the developers, therefore saving costs.
2. Detects error in early stages.

3. Encourages developers to commit small changes without breaking the whole system.
4. Enables quick feedback.

The requirements for Continuous Integration to work are:

1. A version control system for source code.
2. Frequent commits by developers, preferably several times a day, to find out bugs in early stages.

2.3 Software Testing

Software testing is a wide area by itself and this chapter will cover the testing that is being performed under continuous integration. This includes performing unit tests, integration testing and static testing. It will also cover the costs of detecting software bugs in different stages.

Software testing focuses on detecting defects and failures, so that they can be fixed, improving the quality of the software.

"Software testing is the process of applying metrics to determine product quality. Software testing is the dynamic execution of software and the comparison of the results of that execution against a set of pre-determined criteria" (ABI, May 2002). The software test results can be compared to expected value to determine whether the software is working correctly or not.

It is important to detect software faults as early as possible to minimize the costs used to fix them. As seen from the figure below the costs of fixing a software bug can grow up to 30 times the price if not found early stages of development. Working hours or currency can be used as variables for the table. Example, a fault is found in the post-product release of the software, which adds up to 300 working hours from the previous stages. If this fault was found in the first stage, the costs would have been reduced from 300 working hours to 10, lowering the costs caused by the fault dramatically.

Table 1. Relative costs of software fault fixing based on different stages (ABI, May 2002)

Requirement analysis/ Design	Coding/ Unit resting	Integration and system testing	Beta testing	Post-Production release
1 X	5 X	10 X	15 X	30 X

According to ABI, most of the faults are introduced in the requirements gathering and analysis designs (see table 2 below).The bottom row shows the percentage of the faults that are discovered in the specific stage. The “Total” column on the right shows the percentage of where the faults were introduced. The table indicates that 70% of the faults are introduced as early as the first stage of the cycle, but over half are found in the integration stage.

Table 2. A table of where faults are found and where they are introduced in software development stages (ABI, May 2002)

	Where errors are found (%)					
Where errors are introduced (%)	Requirement analysis/ Design	Coding/ Unit resting	Integration and system testing	Beta testing	Post-Production release	Total
Requirement analysis/ Design	3,5	20,5	35	6	15	70
Coding/ Unit resting		6	9	2	3	20
Integration and system testing			6,5	1	2,5	10
Total	3,5	16,5	50,5	9	20,5	100

2.3.1 Unit testing

The objective of unit testing is to take a smallest piece of the software, exclude it from the remaining code and test if the unit works as expected.Each unit is tested separately before merging them. Unit testing is white box testing and is normally performed by the software developers alongside with the actual software development. Unit tests are written in 3 parts:

1. Arrange, here the variables and classes are introduced for the specific tests including the expected value.
2. Act, here the units (functions) are called and parameters from the arrange are chosen.
3. Assert, here the an assert function is called to determine, whether the value gotten from the act is the same as the expected value or not.

These tasks can be coded using a testing framework and they can be made run automatically, which is test automation.

For an example if a login function is unit tested, several tests have to be implemented and the minimum amount of test required to be performed can be defined by edge cases:

1. Logging in with an invalid username
2. Logging in with an invalid password
3. Logging in with an invalid username and password
4. Logging in with an empty username
5. Logging in with an empty password
6. Logging in with an empty username and password
7. Logging in with correct username and password

Each result should be asserted as a fail except for the last case. This should be enough to determine that the login function is working correctly.

2.3.2 Integration testing

“Integration testing is a logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested” (MS Integration Testing 2017). Integration testing reveals the faults that occure when units are combined together. Integration testing is done by software testers using different test scenarios and cases. A test case contain the target being tested, the prerequirements, input data and expected results.

The three most common integration testing strategies are:

1. Top-down approach, which requires the highest-level modules be tested and integrated first. Top-down approach allows high level logic and data flow to be tested early in the process. However, low-level utilities are tested relatively late in the development cycle. Another disadvantage of top-down integration testing is its poor support for early release of limited functionality. (MS Integration Testing 2017).
2. The bottom-up approach requires the lowest-level units be tested and integrated first. By using this approach, utility modules are tested early in the development process. The downside is that the need for drivers complicates test management

and high-level logic and data flow are tested late. Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality. (MS Integration Testing 2017).

3. The third approach, sometimes referred to as the umbrella approach, requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. The potential weaknesses of this approach are significant, however, in that it can be less systematic than the other two approaches, leading to the need for more regression testing. (MS Integration Testing 2017).

2.3.3 Static testing

Static testing is a method where the code is tested without executing it. It is separated into reviewing and static analysis. Reviewing is used to find and fix errors in such as requirement analysis and test cases. There are four types of reviews which are informal, walkthrough, peer review and inspection. The formality of the reviews scale from low to high in the previous order, informal having the lowest level of formality and inspection the highest. In static analysis, the codes of the developers are analysed for structural faults. Static analysis is usually done by tools that interperate the code and give feedback based on coding standards. The types of faults that static analysis reveals are:

- A variable with an undefined value
- Inconsistent interface between modules and components
- Variables that are declared but never used
- Unreachable or Dead code
- Programming standards violations
- Security vulnerabilities
- Syntax violations

(Tutorialspoint Static testing 2017)

3 TOOLS

This chapter will explain the tools that will be used and that are required to perform Continuous Integration and Test Automation for a Laravel project, which is PHP framework.

3.1 Continuous Integration tools

Continuous Integration tools, oftently known as testing servers are the brain component of continuous integration cycle. They are the system that keep polling the source code management system for changes. When a newer version is available, the CI tools triggers a build and the actions the build performes can be configured. The build can include actions like setting up the software, running unit tests, performing static analysis and the list goes on.

CI tools can be either hosted or standalone. The free hosted testing servers have limitations. Free versions are usually limited by building minutes/month, available containers and concurrent builds. The benefits of having a hosted solutions are easier configuration, due to the hosted server having all the tools installed and more friendlier interfaces. The security of hosted services is more reliable, since the installation and security is handled by the company hosting the server. Cloudbees, CodeShip and CircleCI are one of the few examples of hosted CI services.

Standalone solutions are on the other hand free of charge and are only limited by the hardware it is installed on. The cons in a standalone solution is that the whole system needs to be installed and configured by someone in the company. This requires a lot knowledge about security since unauthorized access to the server can lead to unwanted actions such as getting access to the source code and more. Jenkins, Teamcity and Atlassian Bamboo are a few examples of a standalone CI.

3.1.1 Jenkins CI

Jenkins is a Continuous Integration tool written in Java and it is open source, meaning that the source code is available to be studied, edited and improved by developers (Berg

2012). Jenkins has over 1000 plugins, which help the server communicate with wide range of systems. The advantages of using Jenkins are:

- Proven technology that is deployed at a large scale in many organizations
- The code is open source and doesn't have any licensing costs
- It has a web based graphical user interface, which helps setting up jobs, inproves concistency and decreses the maintenance costs
- It is a master slave typology which means that the building and testing is distributed over slave servers. This ensure an environment that is scalable, stable and responsive
- Jenkins supports other languages than just Java
- Jenkins rises the code quality by running tests automatically after committing changes and informing the developers if a build fails

(Berg 2012)

3.2 Test Automation tools

Manual testing is the lowest level of software there is. It is performed by a human sitting infront of a monitor trying out the software with different inputs and cases comparing the results seen on the screen to the expected results. The graphical user interface (frontend) is compared to the layouts made by the graphical designers and the code itself (backend) such as functions are normally compared by giving input, performing a functionality and comparing the results (see the cases under unit testing chapter). Manual tests have to be repeated often during the project, because editing code can break functionalities that previously worked. In order to ensure the responsivity of the software, same tests have to be done onmultiple platforms. This will increase the time used to test and therefore the costs of the testing process. Test automation tools enable to record or write actions that would normally be excecuted manually, compare the results to the expected values and report to the developers wether the tests pass or not. Once the tests are written, they can be ran by a single click. (Smartbear 2017)

3.2.1 PHPUnit

“PHPUnit is a programmer-oriented testing framework for PHP. It is an instance of the xUnit architecture for unit testing frameworks.” (PHPUNIT homepage 2017). It was developed by Sebastian Bergmann and the earliest version listed on their website dates all the way back to August 2013. The newest stable version dates to August 2017, which means it is still active and freguently maintained. PHPUnit enables testing the logic of a php software, which is impossible since php code is translated on the server level. This means that php code never reaches the client side also known as the web browser.

3.2.2 Selenium

“Selenium is a suite of tools specifically for automating web browsers” (About Selenium 2017). Selenium origins go back to 2004, California and it was known as “JavaScriptTestRunner”. Selenium IDE is a plugin which is available on Firefox. It allows to record user interactions on the browser and replaying the actions done as many times as desired. It also allows to modify the speed of the script and the script itself through the plugin and the scripts are exportable. The installation process of Selenium IDE is simple requiring only few clicks. Selenium is used to test the graphical user interface, but can also be used to test functionality.

3.3 Static testing tools

Static testing tools are used by developers as part of coding and testing process. The point is give the code or documents as an input for the testing tool instead of running the code itself. These tools help findind faults that can stay invisible till the release of the product. Having a fault in code doesn’t mean that the whole system will not build, but they are hidden in the system making the system less secure or having low performance.

PHP CodeSniffer is a set of two PHP scripts:

1. Detects violations of a denifed coding standard (PSR-2) in JavaScript, PHP and css files.
2. Corrects coding standard violations automatically.

PHP codesniffer ensures that code is written cleanly, methods have secure visibility and that the naming of methods and variables follow the standard. (PHP Codesniffer 2017)

Phploc is a tool to give metrics of the code such as lines of code, amount of classes and the lengths of classes. (phploc 2017)

"PHP Depend can generate a large set of software metrics from a given code base, these values can be used to measure the quality of a software project and they help to identify that parts of an application where a refactoring should be applied" (PDepend 2017). The full list of available metrics provide by pdepend can be found on the referenced page.

Phpcpd, which is short for php copy/paste detecetor detects if lines of duplicate code is used. (phpcpd 2017)

Php codebrowser generates a browsable representation of PHP code where sections with violations found by quality assurance tools such as PHP_CodeSniffer or PHPMD are highlighted. It can be integrated with Jenkins, which can give high value finding the violations. (phpcb 2017)

Phpdox is used to generate documentation in html form from xml documents. This means that the previously mentioned static testing tool results can be summarized in a single document. (phpdox 2017)

4 CASE : CUSTOMER MANAGEMENT SERVICE SOFTWARE

This chapter will be the practical part, beginning by setting up Jenkins on a linux machine, followed by installation of mandatory software/libraries for the server. After the installation the thesis will explain how to configure Jenkins to work with a private GIT repository and this specific project including unit tests for backend. Writing the test cases and scenarios will be excluded due to the workload being too big for a single thesis. The project will be developed far enough so someone else can continue to design and write the tests.

4.1 Commissioner

TheFirma (see their logo in figure 4 below) is a project learning environment for the students of TUAS. Their office is located in Joukahaisenkatu campus, Joukahaisenkatu 1-3. Theirs services include developing websites, requirements analysis, marketing material, testing, consulting and company training.

Figure 3. Logo of theFirma

4.2 Project

I was approached by a teacher during my internship if I was interested in CI as my thesis subject. Since I had no subject yet and CI relating closely to my work assigments (software tester), I decided to accept the project. KITT's customer management service software is a project written with Laravel-framework, which is ment for the development of php applications. Laravel is one of the most used php frameworks having an active community and a fast method to deliver high quality software which are secure. Laravel

supports object oriented programming by having an mvc-architecture, which reparates applications into 3 parts: models, views and controllers. Models are the structural component of the software, having the definitions of classes. Views contain the graphical content that is shown on the monitor to the users. Controllers have the logic of the software including algorythms and calculations done by the software.

The software is ment for the use of KITT's workers, which are project managers and workers. It provides the possibility of adding new customers, faults, devices and printing the customer contracts. It was developed because the old system was out dated and not secure.

4.3 Installation of mandatory tools

It was decided to set up Jenkins on a Local computer located in the office of Thefirma. The decision was made because the project does not require to be delivered to production continuously and it makes it more secure instead of having in on network. Operating system chosen for Jenkins was linux because it is easier to integrate Jenkins on linux environment.

The work began by installing linux from a clean image file (see figure 5 below).

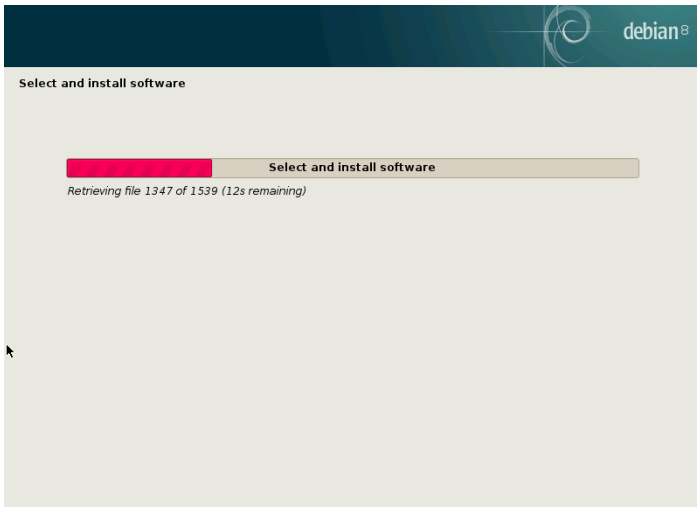

Figure 4. Installation process of Linux

After having a clean install of Linux, it was time to install Java on the machine, since Jenkins is written in java.

4.4 Setting up Jenkings CI

After installing Linux Debian from an AMD 64 image, the first thing to do was install Java higher than 1.8. This was done by adding the following line “deb http://ppa.launchpad.net/webupd8team/java/ubuntu xenial main” to the sources.list file, which is located in /etc/apt/. After editing the file it was time to run apt-get update command.

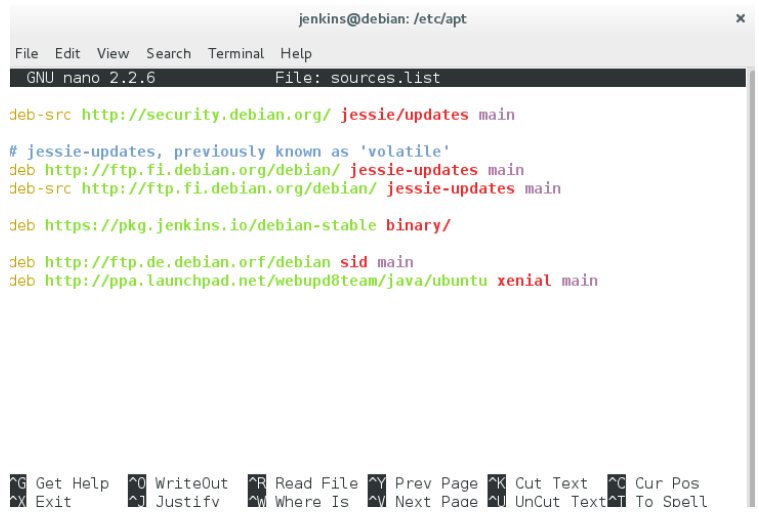

Figure 5. sources.list content

Now java 8 was able to be installed with the following command “apt-get install oracle-java8-installer”. The command will ask wheter to install the packages without verification or no. Terminal will ask to insert the amd64 image while, from which the virtual machine was installed. This can be done from under Devices ->Optical Drives -> IDE (see figure 7).

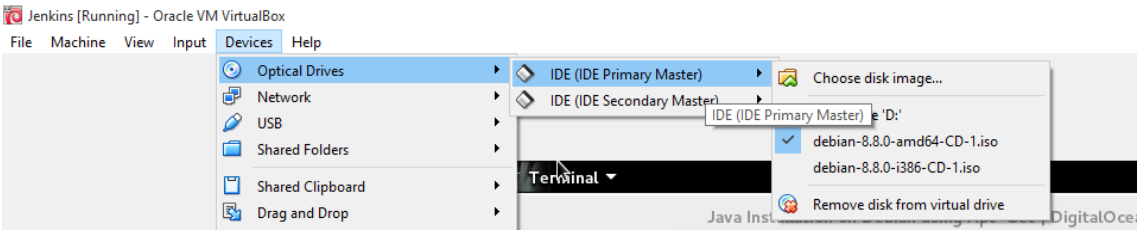

Figure 6. Adding the amd image file

After adding the drive and pressing enter, java started installing automatically. After installation, java version can be checked with “javac –version”- command.

```
root@debian:/etc/apt# javac -version
javac 1.8.0_144
```

Figure 7 Checking Java version

Now having the required Java version, Jenkins is installable. Jenkins will be installed with the following steps:

1. Adding the Jenkins key to the system “wget –q –O – https://pkg.jenkins.io/debian-stable/jenkins.io.key | apt-key add -“
2. Adding Jenkins repository to the sources list “nano /etc/apt/sources.list”, copy paste “deb https://pkg.jenkins.io/debian-stable binary/” and exit
3. Update system repository “apt-get update”
4. Install Jenkins “apt-get install Jenkins”
   (Kawaguchi 2017)

Jenking will be installed and started automatically after the command. Jenkins is installed on port 8080 and can be accessed with a web browser through localhost:8080. Navigating to Jenkins first time will take to an “Unlock Jenkins” page. It is required to navicate to the file, which is location is written on the page, copying the content and pasting ti to the field.

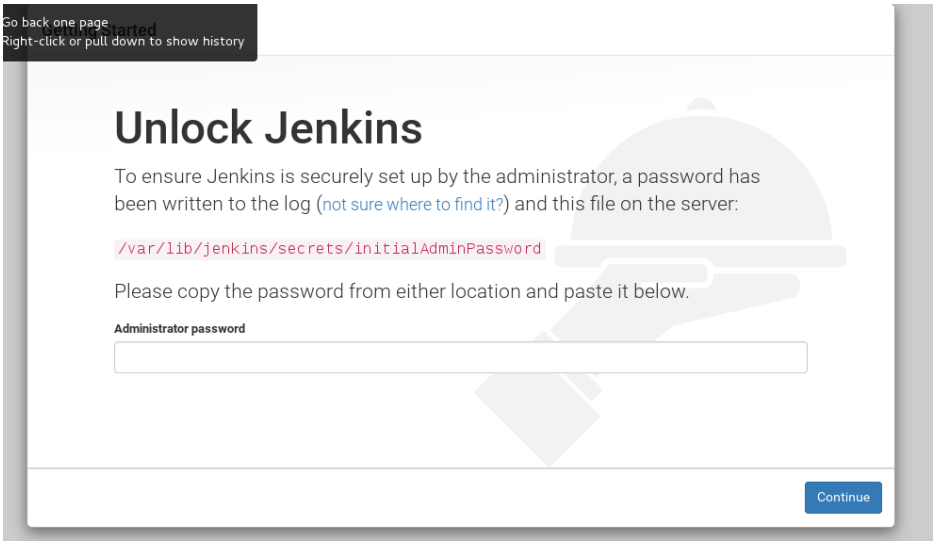

Figure 8. Unlocking Jenkins

Inserting a correct password will redirect the browser to a customization page (see figure 10 below). Since the project is not Java, it requires different plugins. Plugins can always be installed afterwards. Plugins are installed (see figure 11) after choosing them and Jenkins is restared.

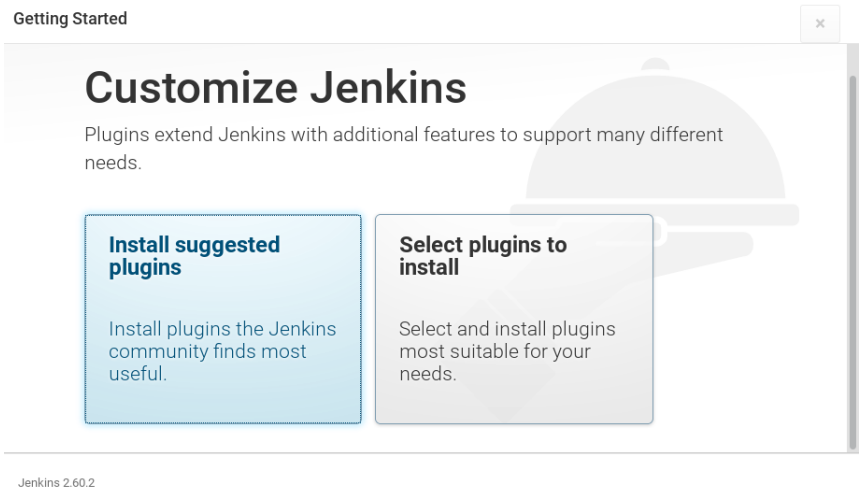

Figure 9. Customization Page

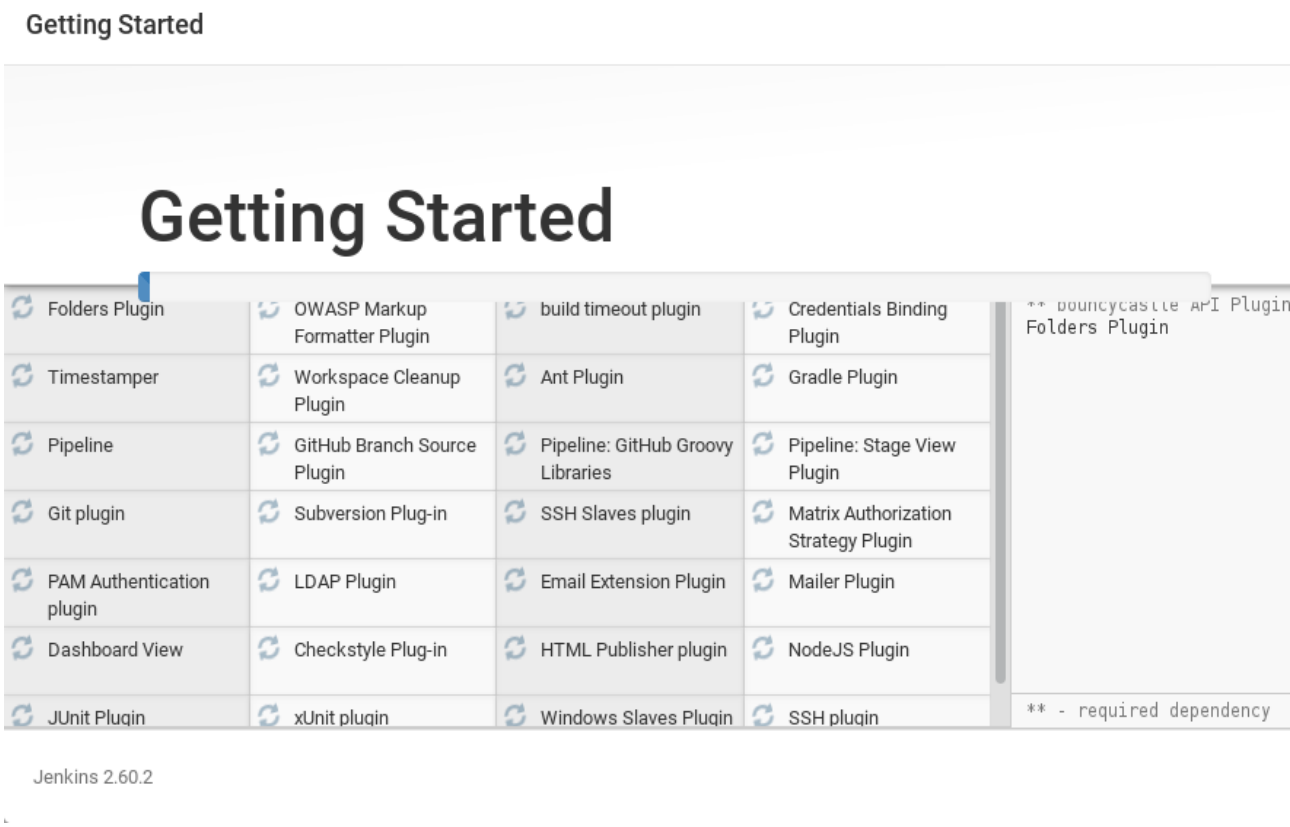

Figure 10. Plugin Installation

User creation follows the plugin installation (highly recommended). After a user is created, the browser redirects to Jenkins dashboard (see figure 12 below).

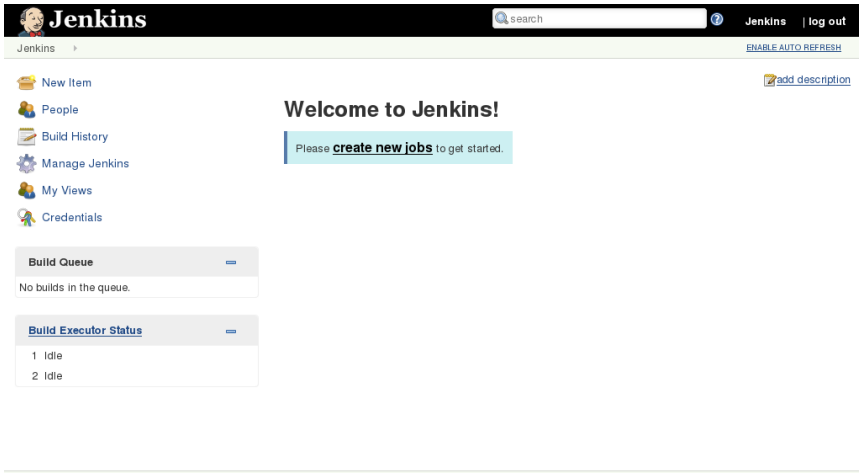

Figure 11. Jenkins dashboard after installation

4.5 Configuring the server and Jenkins for PHP

First of all, the server needs a PHP interpreter. This can be installed on a machine with the following command “apt-get install php5-common libapache2-mod-php5 php5-cli”. Terminal will ask for confirmation and install the filed automatically.

Second thing installed on the machine has to be Composer. Composer is a ment to manage dependency in PHP. Since Laravel Framework supports composer, it speeds up the installation of dependant libraries for the project. Composer is installed in the following steps:

1. Retrieve the setup file with “php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');" command (without the outer quotes).
2. run “php /tmp/composer-setup.php –install-dir=/usr/local/bin –filename=composer”. This will install composer into usr/local/bin, making it available from anywhere on the sytem.

Now having both PHP and composer it was time to install the required PHP tools. These are PHPUnit, PHP Codesniffer, PHPloc, PHP-Depend, PHP Messdetector, PHP Copypastedetector and phpDox (see the chapter under tools -> Static testing tools).

PHPUnit installation is a simple process since it is a PHP Archive (PHAR) and it is done with composer. It is done by adding :

1. Getting the file with “wget https://phar.phpunit.de/phpunit.phar” command
2. making the file executable with “chmod +x phpunit.phar” command
3. moving it to path variable with “mv phpunit.phat /usr/local/bin/phpunit” command

Following these steps will allow the use of PHPUnit anywhere on the system.

Installing PHP Codesniffer can be done using the composer with the following command “ composer global require "squizlabs/php_codesniffer=*" ”, without the outer quotes.

PHPLoc is also a PHP Archive which is installed the following steps:

1. Getting the file with “wget https://phar.phpunit.de/phploc.phar”
2. Making it executable with “chmod +x phploc.phar”
3. moving it to the path variable with “mv phploc.phar /usr/local/bin/phploc”

PDepend:

1. Retrieving the file “wget http://static.pdepend.org/php/latest/pdepend.phar”
2. Making it executable “chmod +x pdepend.phar”
3. moving it to the path variable “mv pdepend.phar /usr/local/bin/pdepend”

PHPmd:

1. “wget –C http://static.phpmd.org/php/latest/phpmd.phar”
2. “chmod +x phpmd.phar”
3. mv phpmd.phar /usr/local/bin/phpmd”

PHP CodeSniffer:

1. apt-get install php-codesniffer

PHP Copy paste detector:

1. wget https://phar.phpunit.de/phpcpd.phar
2. chmod +x phpcpd.phar

3. mv phpcpd.phar /usr/local/bin/phpcpd

PHPdox:

1. “wget http://phpdox.de/releases/phpdox.phar”
2. “chmod +x phpdox.phar”
3. mv phpdox.phar /usr/local/bin/phpdox

Git:

“apt-get install git-core”

Now the machine should have everything installed in order to build a php project. Inorder for Jenkins to have an access to the tools that were installed, Path variable has to be set on Jenkins. This can be done by going to Manage Jenkins -> Configure System -> Global properties, checking “Environment variables” and setting the Name as PATH and the Value as $PATH:vendor/bin:/var/lib/jenkins/.composer/vendor/bin/ (see figure 13 below).

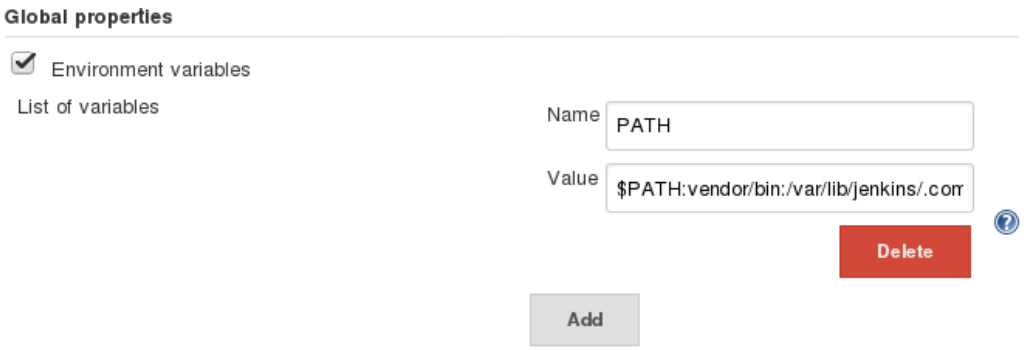

Figure 12. Path variable configuration

Last part is to copy a php template for Jenkins automated build. There is a template developed by Sebastian Bergmann, which creates a build file that is configured to run automated and static tests for a php project. Jenkins templates are located in the jobs folder of Jenkins. The following steps will copy the template correctly:

1. Navigate to Jenkins jobs folder “cd /var/lib/Jenkins/jobs”
2. make a new directory “mkdir php-template”
3. go inside the folder “cd php-template”
4. download the file “wget https://raw.github.com/sebastianbergmann/php-jenkins-template/master/config.xml”
5. go one folder above “cd ..”

6. change the owner of the folder “chown –R Jenkins:Jenkins php-template/”
7. restart Jenkins by navigating to localhost:8080/restart on your browser and clicking “restart”

A new item should appear in the dashboard if the previous steps were done correctly (see figure 14 below)

Figure 13. php-template

4.5.1 Adding a new project

When adding a new project, Jenkins will ask a name and the type of your project. The best suiting option for this case is a freeystyle project since it requires the polling of a version control in order to build the latest version of the software (see figure 15 below).

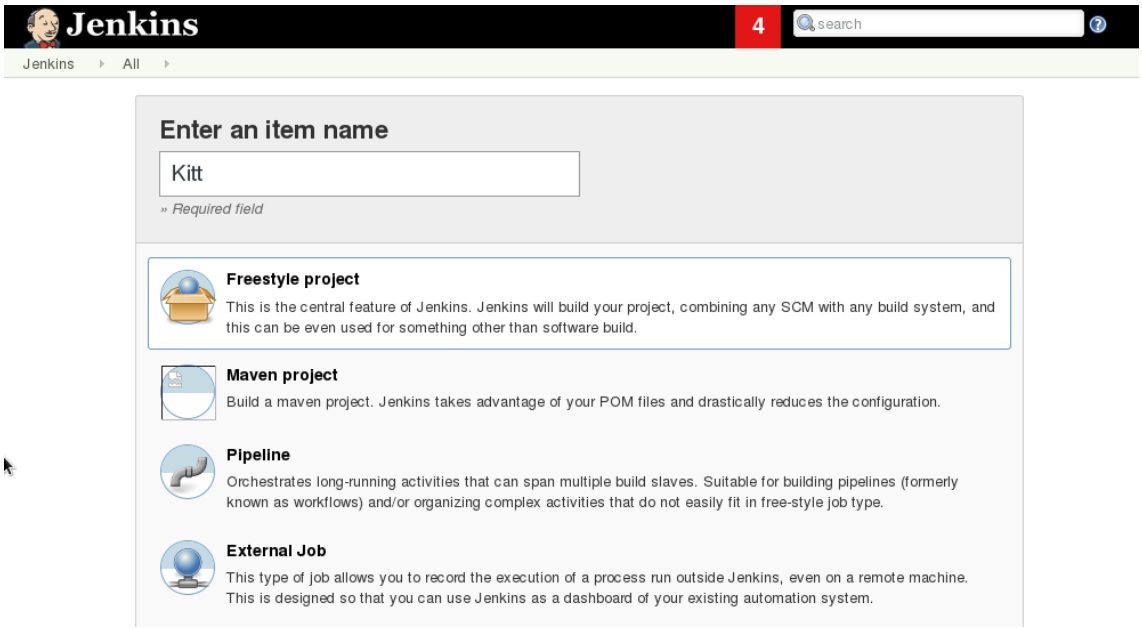

Figure 14 Project name and type selection

It is optional to use a template for a project, but this speeds up the configuration of your project by using a pre determined build file from a template (see figure 16 below).

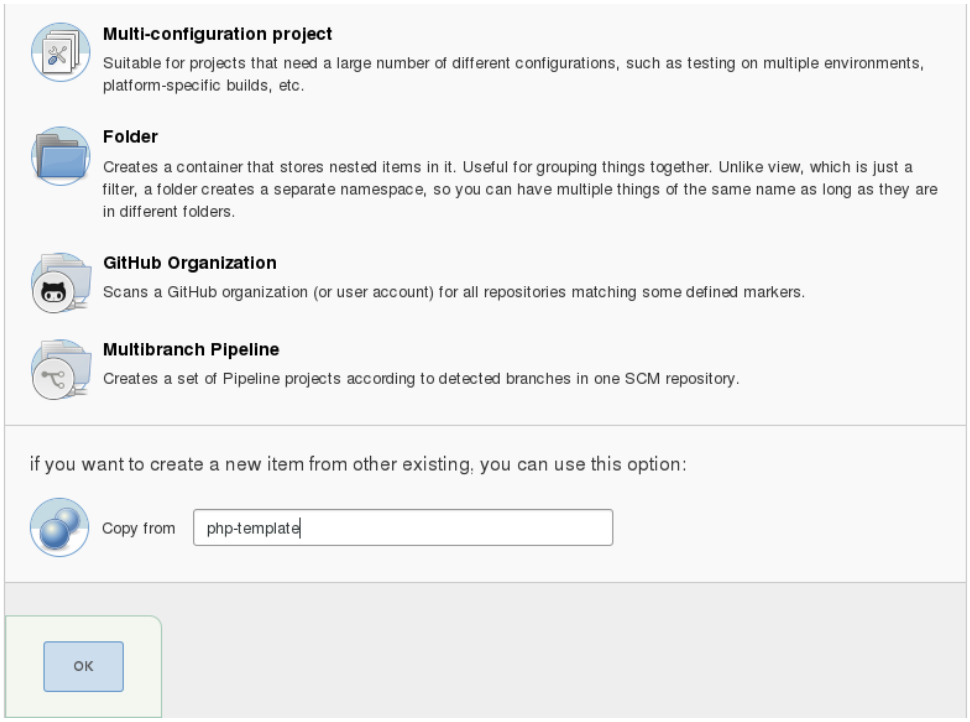

Figure 15. Copy from a template

After pressing the ok button, a new configuration opens up having more detailed settings. The settings are split into 6 different parts which are general, source code management, build triggers, build environment, build and post build actions.The first thing to do is un check “Disable this project” (see figure 17 below).

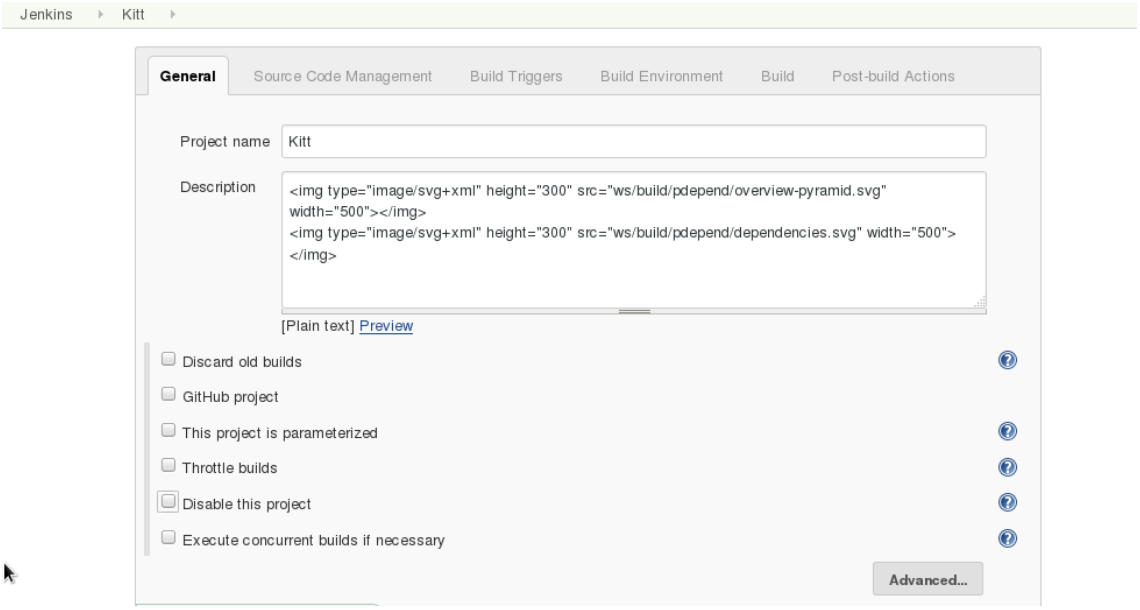

Figure 16. General Settings

One of the most important setting of the project is Source Code Management (see figure 18 below). Here you determine where the project is located. Since the project is located in a private repository, it requires credentials, which can be added during the configuration.

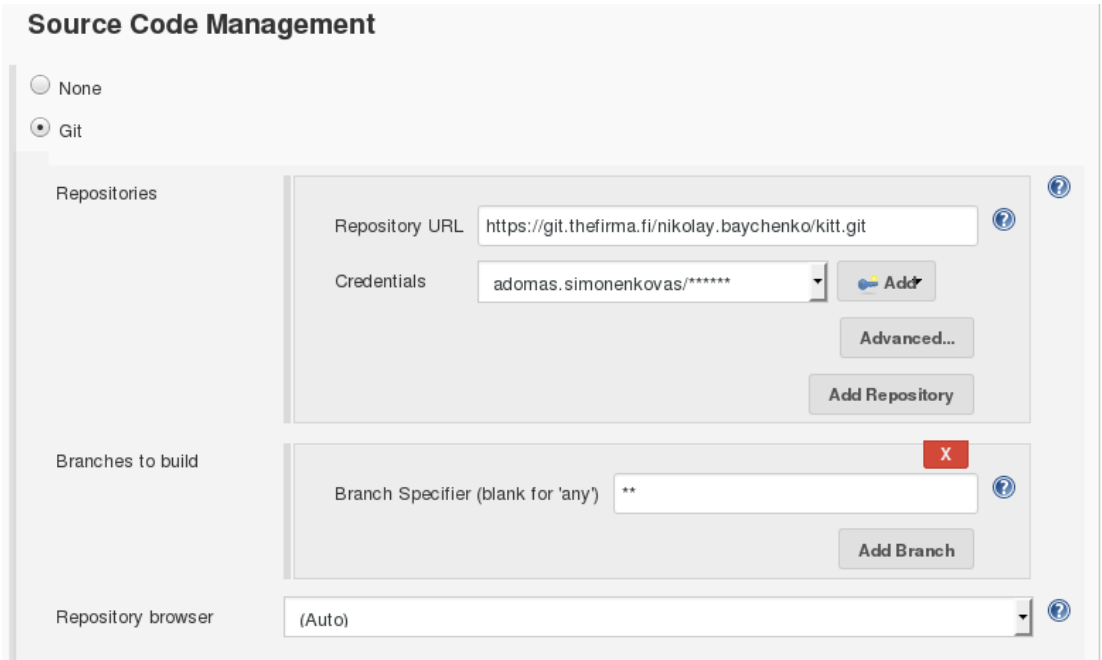

Figure 17. Source Code Management setting

The project should have an automated polling. There are several options in Jenkins for triggering a build. Projects can be build after another project is built, they can be built periodically, a hook can be triggered by GIT to make a build or Jenkins can keep polling the scm for changes and build when a change is detected. Polling the scm was selected for this project, since it is the most rational solution for this case. Five starts in the schedule field indicates that Jenkins will poll the scm every minute (see figure 19 below).

Figure 18. Build Trigger setting

The most important setting is the build. It determines what will be done when a build is triggered. There are different actions Jenkins can do for a build step. The most important steps for this project is executing a shell script and invoking Ant, which means that the build script from the template is ran. Since Laravel runs on composer, the first step should be updating the composer and running install command (see figure 20 below). This will read the composer.json file from the projects root directory and install all dependencies for the project. After everything is installed, a security key is generated for laravels security system.

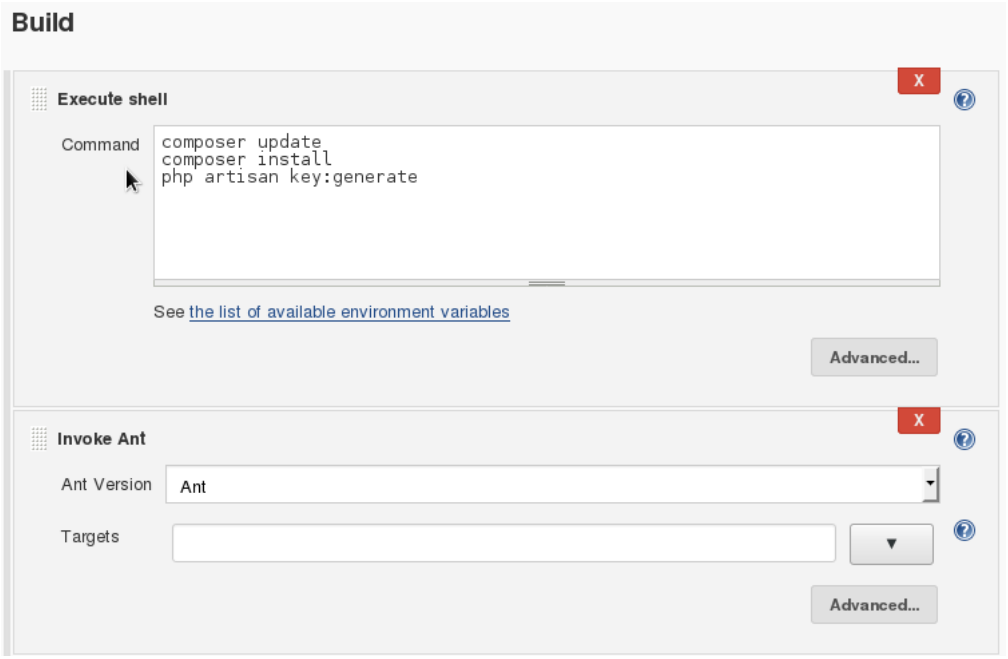

Figure 19. Build step settings

Invoking ant (figure 20) will read the actions from a build file which was copied from the php template.

Post-build actions happen when the build is complete. Post-build actions include publishing static testing results, unit test results and reports containing violations and errors. One of the most important is email notification (see figure 21), which sends the developers an email, if a build is unstable.

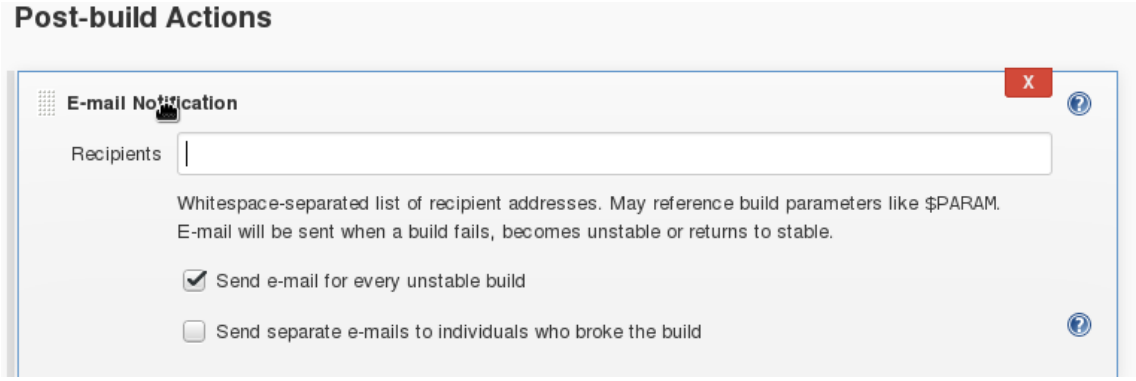

Figure 20. Email notification

After the settings are ready and changes are saved, the browser directs to the project's home page (see Figure 21). Jenkins will run a build a minute after the project is saved

and the status of the build is indicated in build history. A red ball means the build failed, a blue ball means the build is stable, but it can be inproved and a green ball means everything passed.

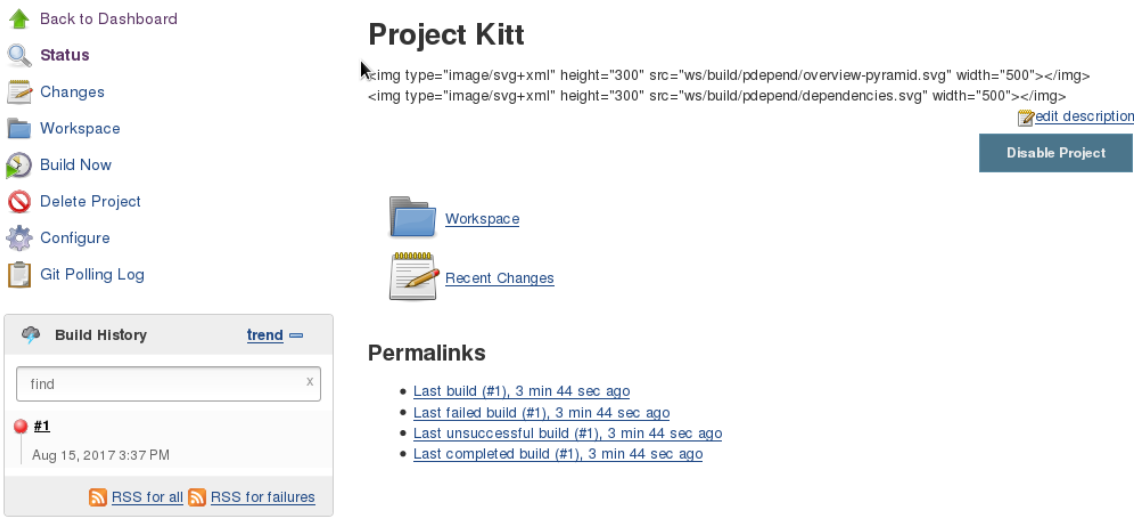

Figure 21. Project home page

Each build have their own information and it can be accessed by clicking the build number located under build history. As seen from the figure 21 below, a change in the SCM triggered this build. Console Output has more information about the build and should tell the reason why a build failed.

Figure 22. Failed build

According to the console output (see figure 23), the first build failed due to a PDO exception indicating a missing driver, which means that mysql not installed on the

machine. It can be done with the following command “apt-get install mysql-server”. This will install and open up the configuration for mysql.

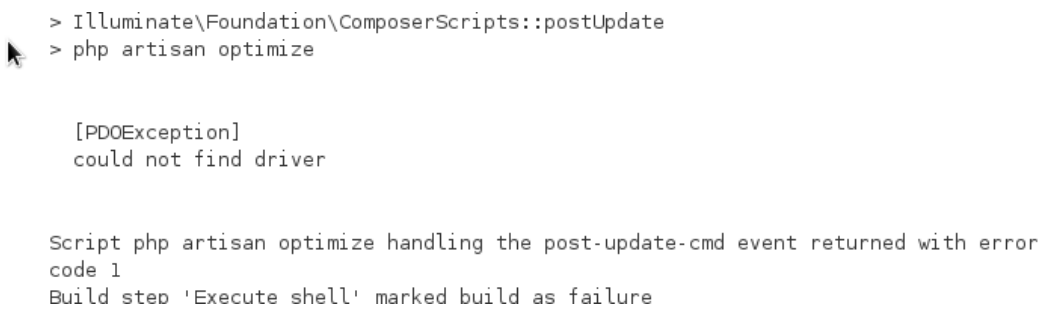

Figure 23. Console output of the first build.

After installing mysql and running the build manually the new build fails again. The new build is failing due to laravel trying to access the database with a user, that is not created (see figure 24 below).

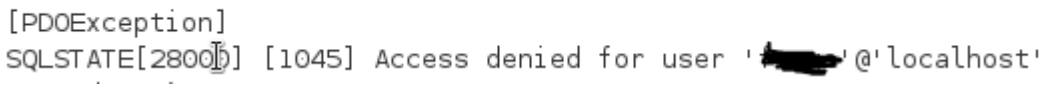

Figure 24. Access denied exception

This problem can be solved by creating a user for a database and granting it all privilages using terminal:

1. mysql –u [your username] –p [your password]
2. CREATE USER ‘[newusername]’@’localhost’; (use data from figure 24)
3. GRANT ALL PRIVILEGES ON * . * TO ‘[newusername]’@’localhost;

The previous problem should go away and now the project should build. A blue ball indicates that the build did not fail, but it has some violations probably in coding standards.

Figure 25. Infromation about the build

All the executed ant targets can be seen on the left side of the screen in the following box:

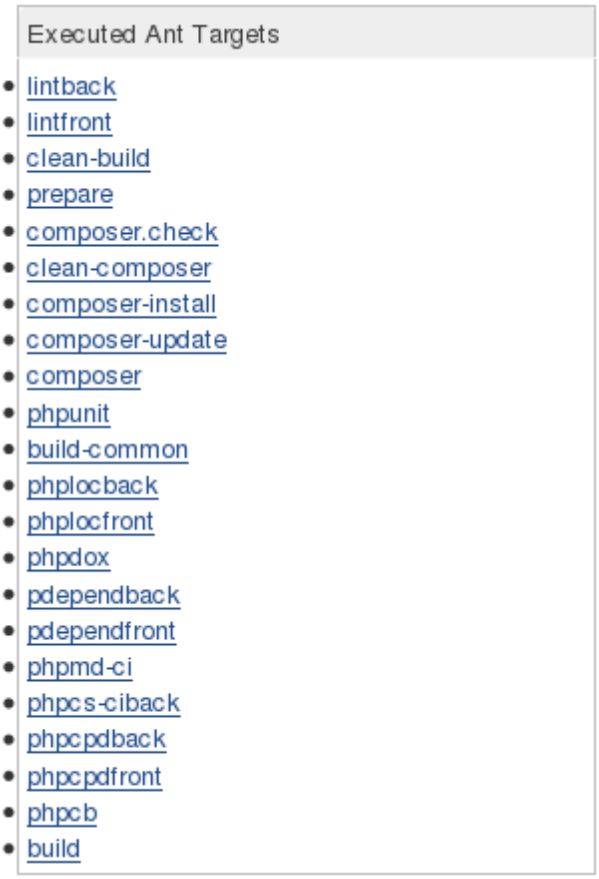

Figure. 26 steps done by Jenkins

The build starts by installing all dependencies of the project. Below in figure 26, installation of components can be seen.

```
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 61 installs, 0 updates, 0 removals
  - Installing symfony/yaml (v3.3.8): Loading from cache
  - Installing sebastian/version (2.0.1): Loading from cache
  - Installing sebastian/resource-operations (1.0.0): Loading from cache
```

Figure 27. Dependency installation

Next step on the build is checking for php syntax errors of the files. Figure 27 below indicates that no syntax errors were found in the files mentioned below.

```
lintback:
    [apply] No syntax errors detected in /var/lib/jenkins/workspace/Kitt/resources
/views/announcement/create.blade.php
    [apply] No syntax errors detected in /var/lib/jenkins/workspace/Kitt/resources
/views/announcement/edit.blade.php
```

Figure 28. Syntax check

After the syntax is checked Jenkins clears and recreates the build directory (see figure 28 below). This ensures that there is no corrupted data:

```
clean-build:
     [echo] Cleaning out the build artifacts
   [delete] Deleting directory /var/lib/jenkins/workspace/Kitt/build/code-browser
   [delete] Deleting directory /var/lib/jenkins/workspace/Kitt/build/coverage
   [delete] Deleting directory /var/lib/jenkins/workspace/Kitt/build/logs
   [delete] Deleting directory /var/lib/jenkins/workspace/Kitt/build/pdepend
```

Figure 29. Deleting up the build directory

Folders are re created after the deletion.

```
prepare:
     [echo] Making the build artifact folders
    [mkdir] Created dir: /var/lib/jenkins/workspace/Kitt/build/api
    [mkdir] Created dir: /var/lib/jenkins/workspace/Kitt/build/code-browser
    [mkdir] Created dir: /var/lib/jenkins/workspace/Kitt/build/coverage
    [mkdir] Created dir: /var/lib/jenkins/workspace/Kitt/build/logs
    [mkdir] Created dir: /var/lib/jenkins/workspace/Kitt/build/pdepend
```

Figure 30. Re creating build folders

Unit tests are run first, but since this thesis only implemented Jenkins to notice the tests, but not write the tests. Figure 30 below shows that 1 test ran and that the assertion was

true. The test was a simple assertion and the point of the test was to see if it is ran automatically.

```
phpunit:
     [exec] PHPUnit 5.7.0 by Sebastian Bergmann and contributors.
     [exec]
     [exec] Error:         No code coverage driver is available
     [exec]
     [exec] .
/ 1 (100%)
     [exec]
     [exec] Time: 47 ms, Memory: 4.50MB
     [exec]
     [exec] OK (1 test, 1 assertion)
```

Figure 31. Running unit tests

Static test show some metrics of predefined files. Since in Laravel the code is separated in different parts, frontend and backend needs to be tested in different tasks.

```
phplocback:
     [exec] phploc 4.0.0 by Sebastian Bergmann.
     [exec]
     [exec] Directories                                  11
     [exec] Files                                        50
     [exec]
     [exec] Size
     [exec]   Lines of Code (LOC)                      3954
     [exec]   Comment Lines of Code (CLOC)             1388 (35.10%)
     [exec]   Non-Comment Lines of Code (NCLOC)        2566 (64.90%)
     [exec]   Logical Lines of Code (LLOC)              741 (18.74%)
     [exec]     Classes                                 496 (66.94%)
     [exec]       Average Class Length                   10
     [exec]         Minimum Class Length                  0
     [exec]         Maximum Class Length                 53
     [exec]       Average Method Length                   2
     [exec]         Minimum Method Length                 0
     [exec]         Maximum Method Length                12
     [exec]     Functions                                 3 (0.40%)
     [exec]       Average Function Length                 0
     [exec]     Not in classes or functions             242 (32.66%)
     [exec]
```

Figure 32. Running phploc

```
pdependback:
     [exec] PDepend 2.5.0
     [exec]
     [exec] Parsing source files:
     [exec] ..................................................      50
     [exec]
     [exec] Calculating Dependency metrics:
     [exec] ...........                                             234
     [exec]
     [exec] Calculating Coupling metrics:
     [exec] ................                                        340
     [exec]
     [exec] Calculating Cyclomatic Complexity metrics:
     [exec] ................                                        340
     [exec]
     [exec] Calculating Inheritance metrics:
     [exec] ...                                                      70
     [exec]
     [exec] Calculating Node Count metrics:
     [exec] ...........                                             237
     [exec]
     [exec] Calculating Node Loc metrics:
     [exec] ..............                                          286
     [exec]
     [exec] Generating pdepend log files, this may take a moment.
     [exec]
     [exec] Time: 0:00:03; Memory: 18.50Mb
```

Figure 33. running PDepend

Figure 33 below shows the results of phpcpd for the backend code of the project. As seen from the picture there are no duplicated lines of code in the files.

```
phpcpdback:
     [exec] phpcpd 3.0.0 by Sebastian Bergmann.
     [exec]
     [exec] 0.00% duplicated lines out of 3954 total lines of code.
     [exec]
     [exec] Time: 105 ms, Memory: 5.25MB
```

Figure 34. Results of phpcpdback task

As seen from figure 34 below, the build was successful and the software passed all the tests that were set at the configuration step.

```
build:

BUILD SUCCESSFUL
Total time: 20 seconds
Finished: SUCCESS
```

Figure 35. build information

5 CONCLUSION

The purpose of this thesis was to find out and implement the best suitable solution for the project being tested, which was a Php application. It was decided to implement it on a local machine located in the office of TheFirma, since the project did not require automatical deployment. As having prior knowledge of developing php applications, It was known that a php translator and composer was required on the server. Research revealed that continuous integration is really simple, fast to set up and simplifies the process of software testing during the software development life cycle. It has open source build solutions for different applications and can be modified widely to meet any company's requirements

The results reveal that even though the application may build to a working version, it may contain a lot of violations that effect the performance and security of the application. CI could save software development companies costs resulted due to software bugs.

The amount of information learned writing this thesis was umbelievably big. During the research I learned how to search reliable sources, how to find the relevant information from them and some general information about software testing, version control, test automation tools and continuous integration. During the installation of the server I learned both windows and linux as a server environment. Linux was chosen because of some tools could not be installed on a windows environment. Reading the code taught me the functionality of Laravel framework.

This thesis did not implement the dynamic tests on the project due to the workload of designing test scenarios and cases for the project. I highly recommended this project to be continued by developing tests and using the created system to run the tests and analyze the results.

REFERENCES

Hughey, D. (2009). Available at: http://www.umsl.edu/~hugheyd/is6840/waterfall.html

Berg, A. (2013) Jenkins Continuous Integration Cookbook

What is Version Control. Available at: https://www.atlassian.com/git/tutorials/what-is-version-control

Getting Started - About Version Control. Available at: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Karadzhov, S (2014). Fundamental of Continuous Integration with Jenkins and Zend Serverl. Available at: http://static.zend.com/topics/WP-Fundamentals-of-Continuous-Integration-with-Jenkins-and-Zend-Server-2014-03-31-EN.pdf

Figure 2 James, E (2013) https://softwareengineering.stackexchange.com/questions/149020/good-workflow-for-software-development-with-scrum-and-continuous-integration

ABI May (2002). Available at: https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf

MS Unit testing. Available at: https://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx

MS Integration testing. Available at: https://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx)

Tutorialspoint static testing (2017). Available at: https://www.tutorialspoint.com/software_testing_dictionary/static_testing.htm

Smartbear Automated Testing (2017). Available at: https://smartbear.com/learn/automated-testing

PHPUnit homepage. Available at: https://phpunit.de/index.html

About Selenium. Available at: http://www.seleniumhq.org/about

PHP Codesniffer. Available at: https://pear.php.net/package/PHP_CodeSniffer

phploc (2017). Available at: https://github.com/sebastianbergmann/phploc

Pdepend (2017). Available at: https://pdepend.org/documentation/software-metrics/index.html

phpcpd (2017). Available at: https://github.com/sebastianbergmann/phpcpd

phpcb (2017). Available at: https://github.com/mayflower/PHP_CodeBrowser

phpdox (2017). Available at: http://phpdox.de/index.html

Kawaguchi, K. (2017) Installing Jenkins on Ubuntu. Available at: https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+on+Ubuntu